

리니지 유저 이탈 예측 모형

우리팀화이팅

강재훈 명재성 안주영 정민지 정지혜



1. 문제 정의
2. 데이터 이해 & EDA
3. 데이터 전처리 & Feature Engineering
4. 모델링
5. 결과 해석

01. 문제 정의

01

02

03

04

05

리니지 고객(유저) 활동 데이터를 활용하여 잔존 가치를 고려한 이탈 예측 모형 개발

분석 문제

유저 이탈 여부
유저 생존 기간 (이탈 시기)
유저의 생존 기간에 따른 평균 결제 금액
(고객별 예상 매출)

➤➤ 고객의 기대이익을 고려한 모형 구축

이탈 기준

64일 동안 이탈하지 않으면 ➔ 잔존

64일 전에 이탈하면 ➔ 이탈

잔존가치의 합이 최대가 되도록 고객의 생존기간과 결제액을 예측

02. 데이터 이해 & EDA

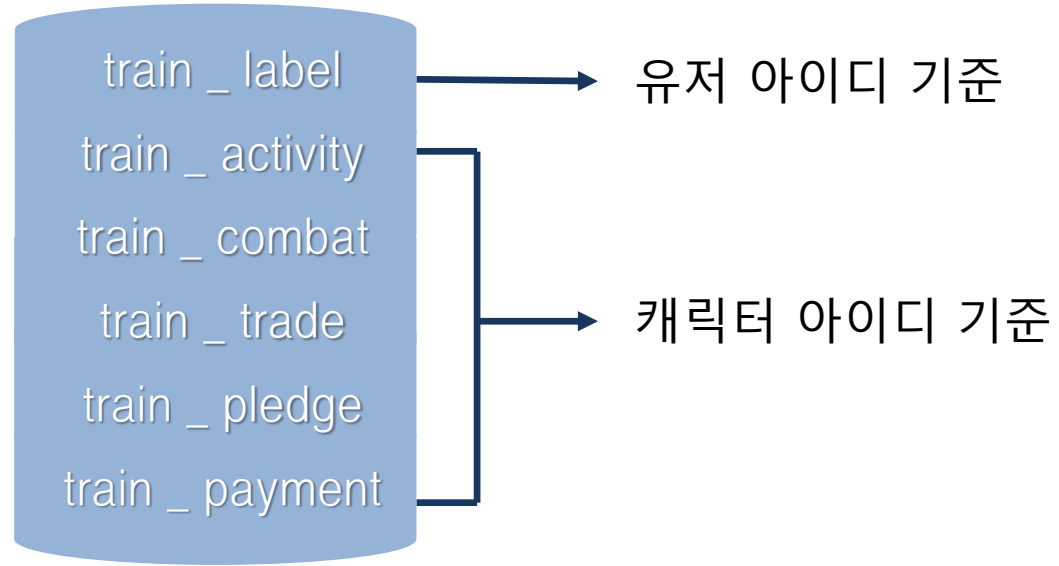
01

02

03

04

05



주어진 데이터는 캐릭터의 각 항목별 일일 활동 집계이며

한 명의 유저가 갖는 캐릭터는 하나 이상이다

그러나, 예측해야 할 대상은 유저의 생존 시간과 그 기간에 따른 일 평균 결제 금액이므로

하나의 유저가 갖는 캐릭터의 변수들을 평균하거나 합산하여 유저기준으로 정렬한다

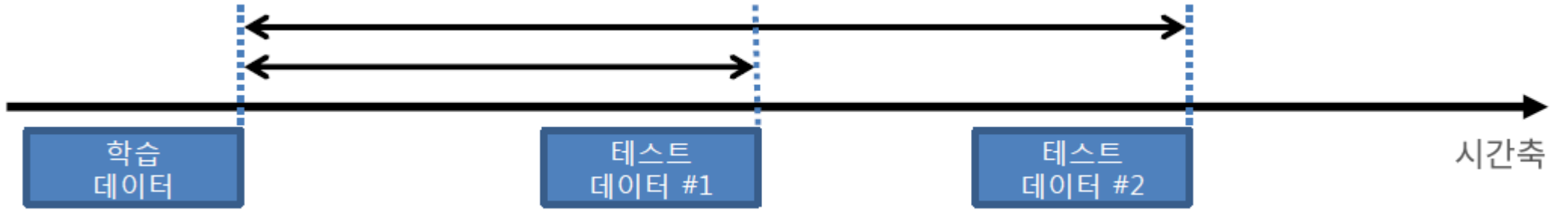
01

02

03

04

05



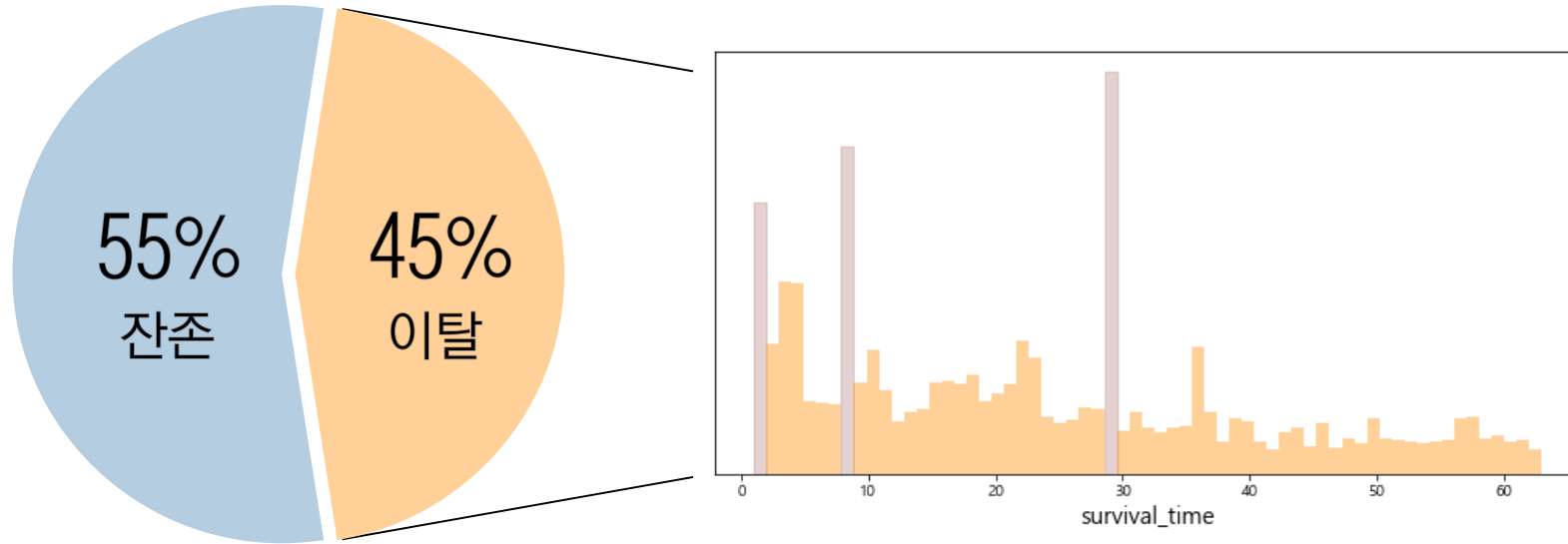
제공된 train, test1, test2 데이터는 각각 수집된 시점이 다른 time-shift된 특성을 가진다

train 데이터에 오버피팅되면 오히려 test1, test2 데이터의 시점에는 맞지않을 수 있다

최대한 시간의 변화에 강건한 모델을 만들어야 한다

01

»» 생존 시간 : survival_time



잔존과 이탈 고객의 비율은 balanced하다

생존시간으로 보면 1일에서 64일 중 64일이 전체 중에서 55%를 차지한다

이탈 고객 중에서도 생존기간이 1, 8, 29일인 이탈자가 많다

01

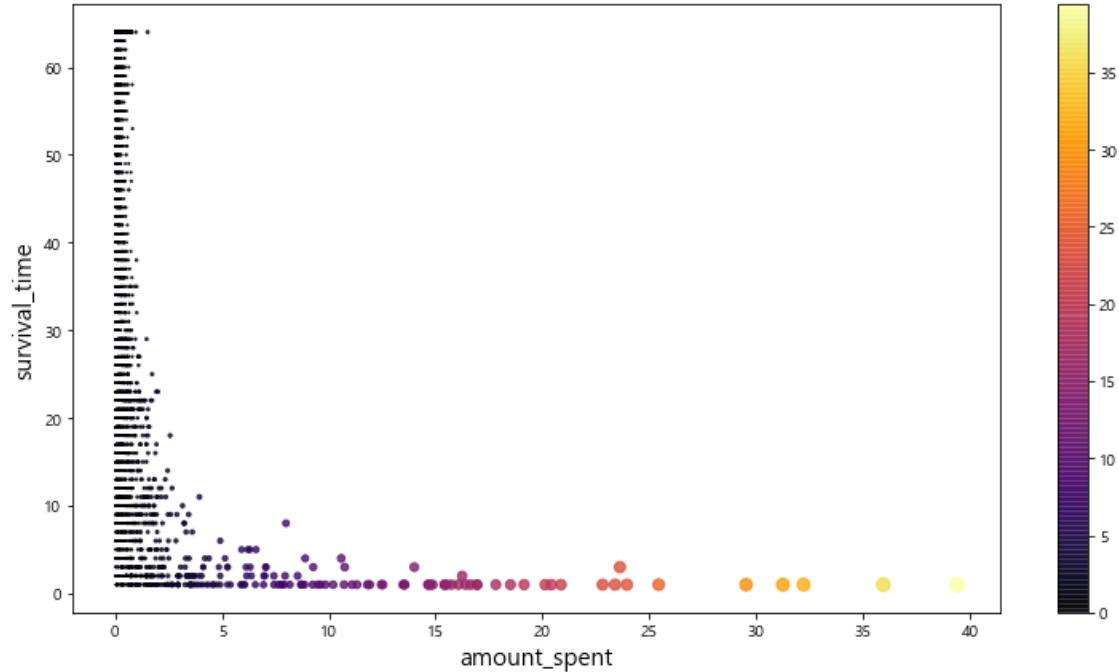
»» 평균 결제 금액 : amount_spent

02

03

04

05



결제 금액이 0인 유저는 59%로 절반 이상의 유저가 과금을 하지 않았다

실제로 큰 금액을 결제한 유저는 이탈한다

하지만 기대이익을 크게 하기 위해서 큰 금액을 결제 하는 유저가 이탈하지 않는 것이 중요하다

01

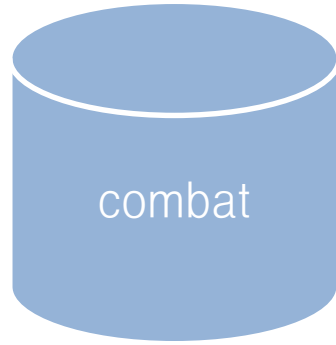
>>> Combat

02

03

04

05



pledge_cnt	random_attacker_cnt	random_defender_cnt	temp_cnt	same_pledge_cnt	etc_cnt	num_opponent
0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000

제공된 combat 데이터를 보면, 전투 활동 변수들의 값이 0인 경우가 매우 많다
하나의 전투 활동 변수라도 0보다 큰 row를 실질적으로 전투 활동을 한 경우로 했을 때,

전체 중 28%의 row만이 실질적인 전투 활동 데이터

01

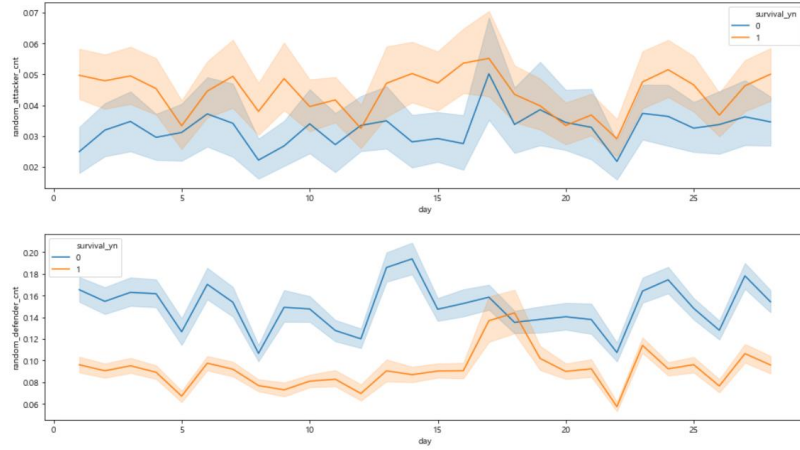
02

03

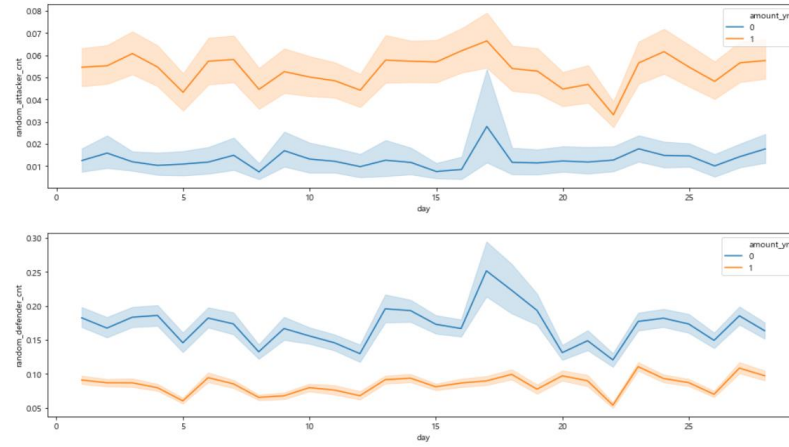
04

05

Combat



< 이탈 / 잔존 >



< 과금 / 무과금 >



대부분의 전투활동 변수에서 잔존하는 사람들과 과금하는 사람들은 값이 크다

01

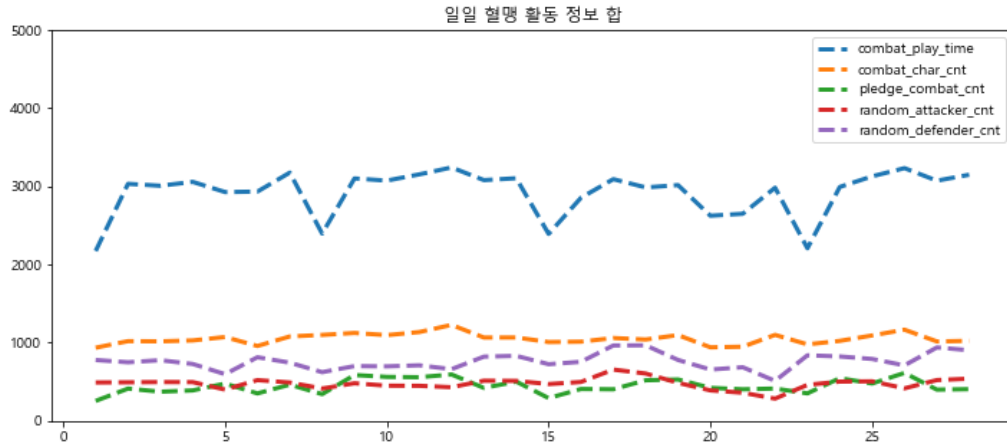
02

03

04

05

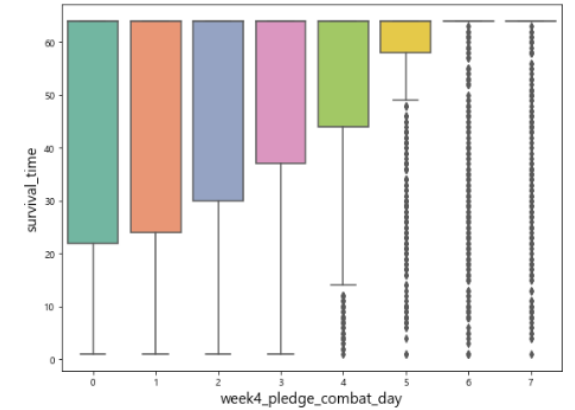
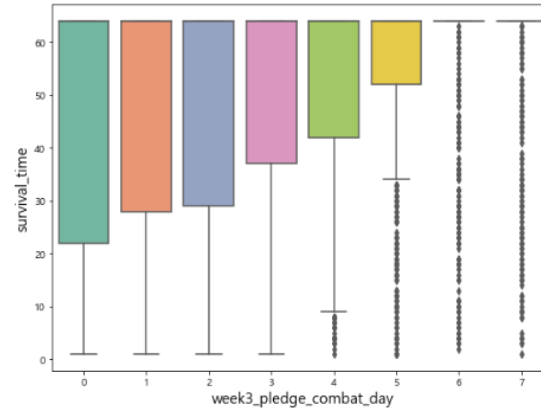
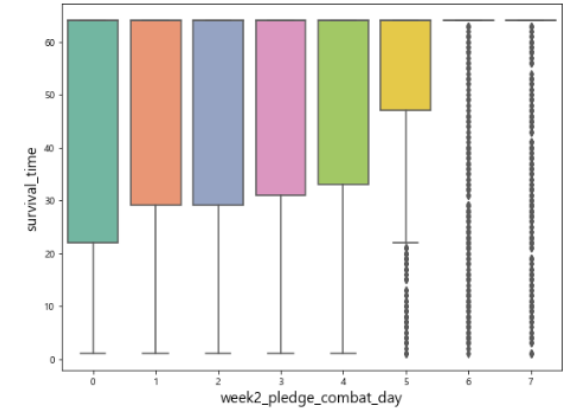
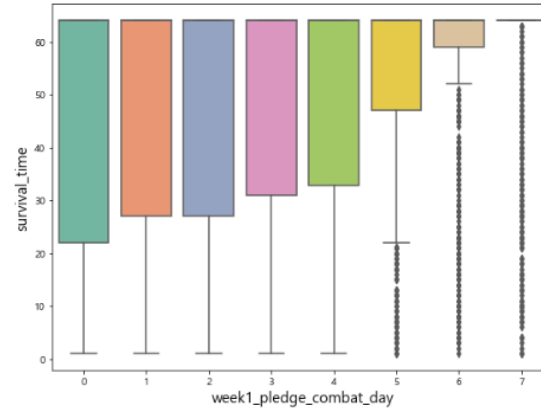
»» Pledge



혈맹 전투 활동 시간은 주단위로 비슷한 패턴을 보인다

주차별로 유저 혈맹 전투 참여 일수를 보면,

혈맹 전투가 잦은 유저들의 생존 시간이 대체적으로 크다



03.
데이터 전처리
& Feature Engineering

01

02

03

»» 파생 변수 생성

activity

- **활동**
 - num_of_characters 28일 동안 플레이한 캐릭터 수의 합
 - num_of_days 28일간 게임 접속 일수
 - num_of_servers 28일 동안 플레이한 서버 수
 - act_first_day acc_id가 처음으로 플레이한 day
 - act_first_day_diff 처음으로 접속한 day와 마지막 day의 차 (마지막날 모두 day 28)
 - num_of_game_money_change 아데나 보유량의 변동 횟수

04

05

payment

- **빈도** pay_day_cnt 28일 동안 유저가 결제한 날짜 수

01

combat

02

03

04

05

- Day

 - combat_active_days 전투활성일수
 - combat_active_days_rate 전투활동일 중 전투활성일의 비율 (= 전투활성일/전투활동일)
 - combat_active_⟨first / last⟩_day 전투 활성 data 기준 ⟨첫번째 / 마지막⟩ 접속일

- 캐릭터

 - ⟨ class / char ⟩_cnt 보유한 ⟨클래스 종류 / 캐릭터⟩ 개수
 - total_⟨min / max⟩_level 보유한 캐릭터의 레벨 중 가장 ⟨낮은 / 높은⟩ 레벨
 - ⟨ 클래스 ⟩_rate 보유한 클래스 중 해당 ⟨클래스⟩인 캐릭터 비율
※ 클래스 : 군주, 기사, 요정, 마법사, 다크엘프, 용기사, 환술사, 전사
 - sp_cls_rate 보유한 캐릭터 중 특별한 클래스(마법사, 요정, 다크엘프)인 캐릭터 비율
 - level_group 가장 높은 레벨이 레벨범주 10(50~54)을 기준으로 작은지(0), 속해있는지(1), 더 큰지(2)
 - level_down_char_cnt 보유한 캐릭터 중 level down을 경험한 캐릭터의 수
 - level_up_rate 보유한 캐릭터 중 level up을 경험한 캐릭터의 비율
 - level_keep_rate 보유한 캐릭터 중 level을 유지한 캐릭터의 비율

- 전투 활동

 - act_mean_of_⟨전투활동변수⟩ 실질적으로 전투활동을 한 날의 각 전투활동변수의 평균
= sum_of_전투활동/전투활성일수

01

pledge

02

03

■ 혈맹 가입 현황

pledge_days	혈맹활동일수
pledged_char_cnt	혈맹에 가입한 캐릭터 수
pledge_cnt	유저의 캐릭터가 가입한 혈맹 수
same_pledged_char_cnt	동일 혈맹인 캐릭터 보유 수
plural_pledge_cnt	같은 혈맹에 있는 캐릭터 2개 이상인 혈맹 수
pledge_changed_char_cnt	혈맹 변경 캐릭터 수
pledge_changed_cnt	유저의 총 혈맹 변경 수
week#_pledge_combat_day	주별 혈맹 전투에 참여한 일수
play_server_cnt	이용한 서버 수의 평균
non_combat_pledge_cnt	소속혈맹 중 비전투 혈맹의 수
conflict_pledge_cnt	소속혈맹 중 동일혈맹 내 전투를 경험한 혈맹 수

04

05

■ 혈맹 단위 활동

play_server_cnt	가입한 혈맹이 이용한 서버 수들의 평균
combat_play_char_ratio	가입한 혈맹의 활동 캐릭터 대비 전투참여 캐릭터 비율들의 평균

01

trade

02

03

- Day

last_sell_day / last_buy_day	판매/구매를 진행한 마지막 활동 일
num_sell_day / num_buy_day	판매/구매를 진행한 일(day) 수
last_sell_item_amount / last_buy_item_amount	마지막 활동 일의 아이템 판매/구매량
last_sell_item_price / last_buy_item_price	마지막 활동 일의 아이템 판매/구매 가격
last_sell_item_type / last_buy_item_type	마지막 활동 일에 판매/구매한 아이템 종류

- 거래기록

num_sell / num_buy	총 판매/구매 빈도
num_sell_character / num_buy_character	판매/구매를 진행한 캐릭터 수
num_trade_server	거래를 진행한 서버 수
sell_item_amount / buy_item_amount	평균 아이템 판매/구매량
sell_item_price / buy_item_price	평균 아이템 판매/구매 가격
sell_time / buy_time	판매/구매가 가장 많이 발생한 시간(hour); (-1 : 관측되지 않음)
sell_type / buy_type	판매/구매 시 가장 많이 활용한 거래 종류; (1 : 교환창 / 0 : 개인상점 / -1 : 관측되지 않음)
common_item_sell / common_item_buy	가장 많이 판매/구매한 아이템의 종류

04

05

01

02

»» Flattened Data

03

집계 변수와 거래, 혈맹 변수를 제외하고
1~28일 간의 시계열적인 특성을 반영할 수 있도록 **day별 변수 생성**

04

05

acc_id	day	col1	...
1	1	0.2	
1	2	0	
1	3	1	
1	4	1.5	
...	



acc_id	day1_col1	day2_col1	...	day28_col#
1	0.2	0	...	

만약 1~28일 중 접속하지 않은 day가 있다면 그 날의 값은 **0**으로 대체

01

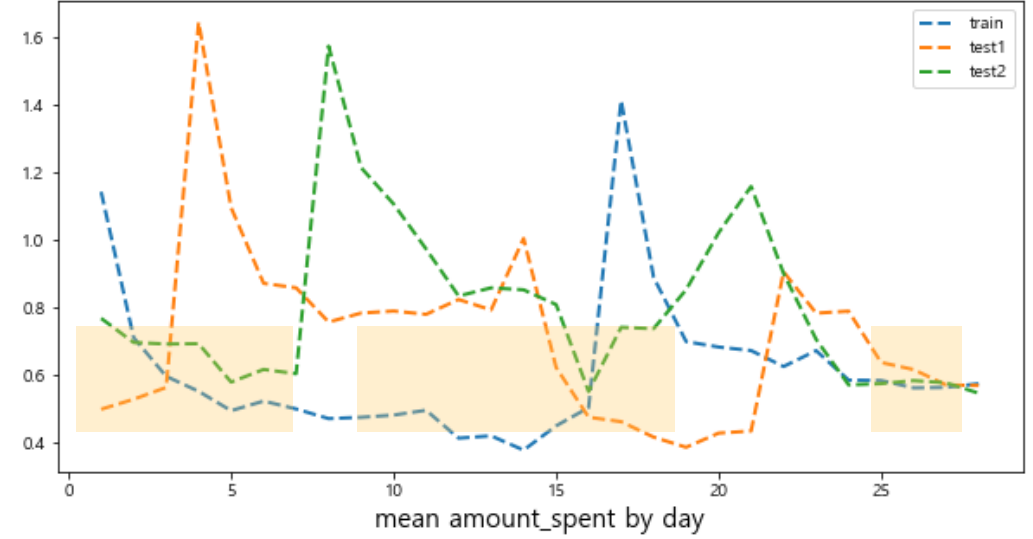
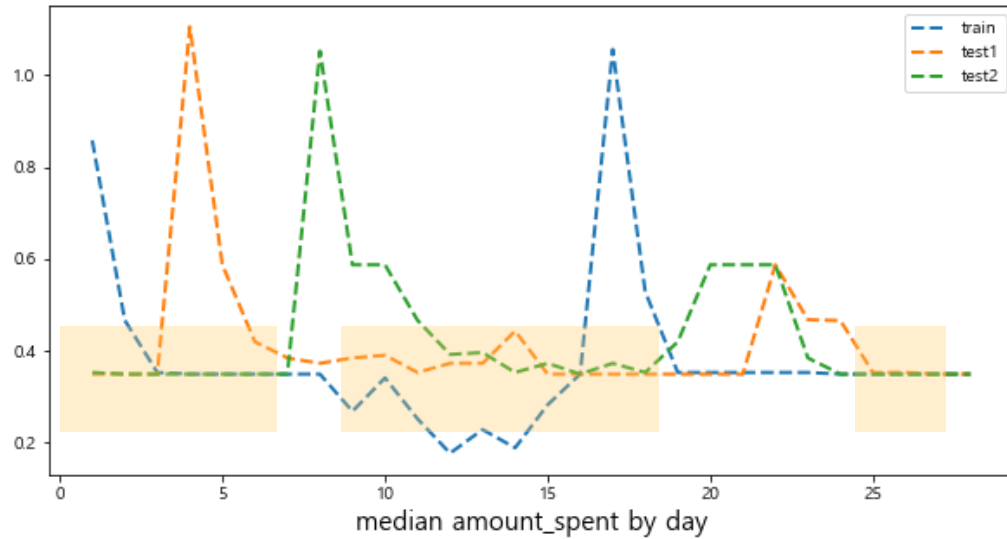
02

03

04

05

Feature Selection



일별 결제금액 평균과 중위값을 보면 train과 test1, 2의 day에 따른 패턴이 다르다

이벤트나 정기 점검일의 이동으로 인한 train과 test의 차이를 줄이기 위해

평균 결제 금액이 안정적인 날들의 변수를 선택한다

01

02

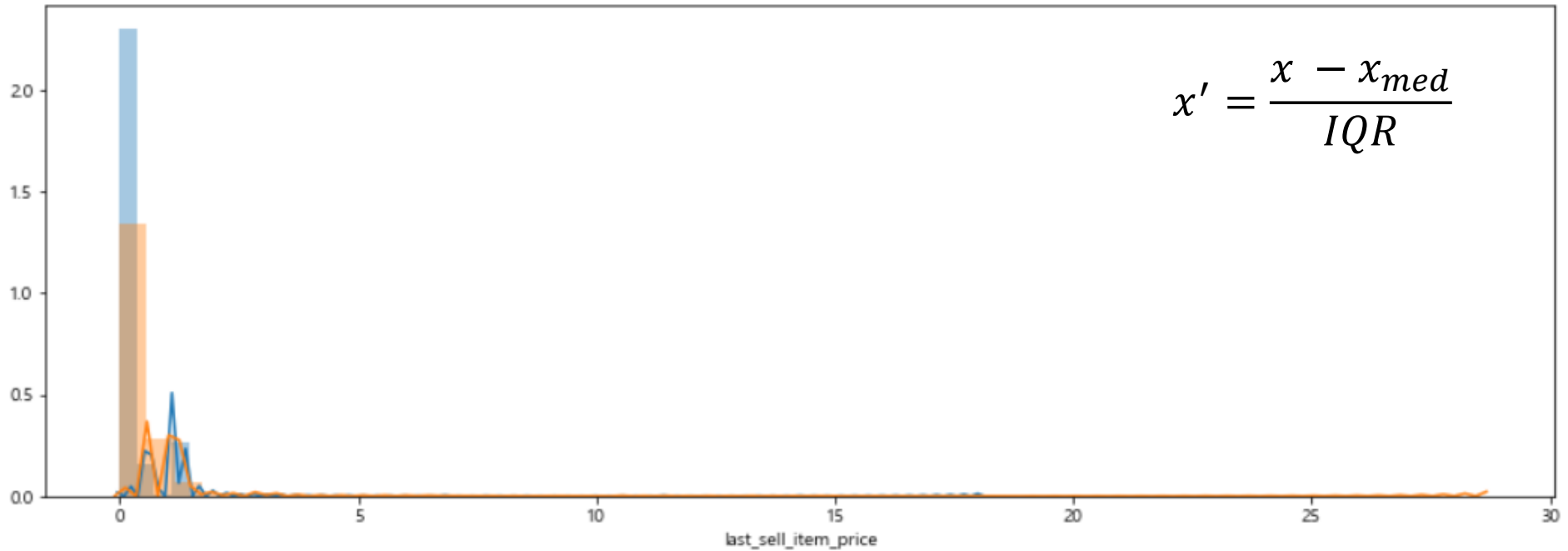
03

04

05

➤➤ Robust Scaling

- 대부분의 feature에 극단적인 outlier 존재한다
- outlier에 영향을 받지않는 robust scaling
- train, test1, test2를 모두 활용하여 scaling 진행



$$x' = \frac{x - x_{med}}{IQR}$$

01

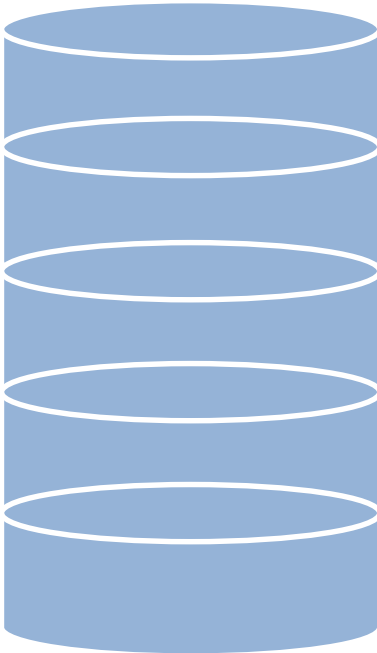
02

03

04

05

Raw data



acc_id 기준으로 주어진 5개의 data를 모두 정리

acc_id	sum_of_<>	...	day1_<>	...
1	0.2		0.6	
2	0		3.4	
3	1		2	
4	1.5		1.8	
...		

04. 모델링

01

02

03

04

05

»» First approach,

Deep Learning Model

»» Second approach,

Tree-based Model

01

02

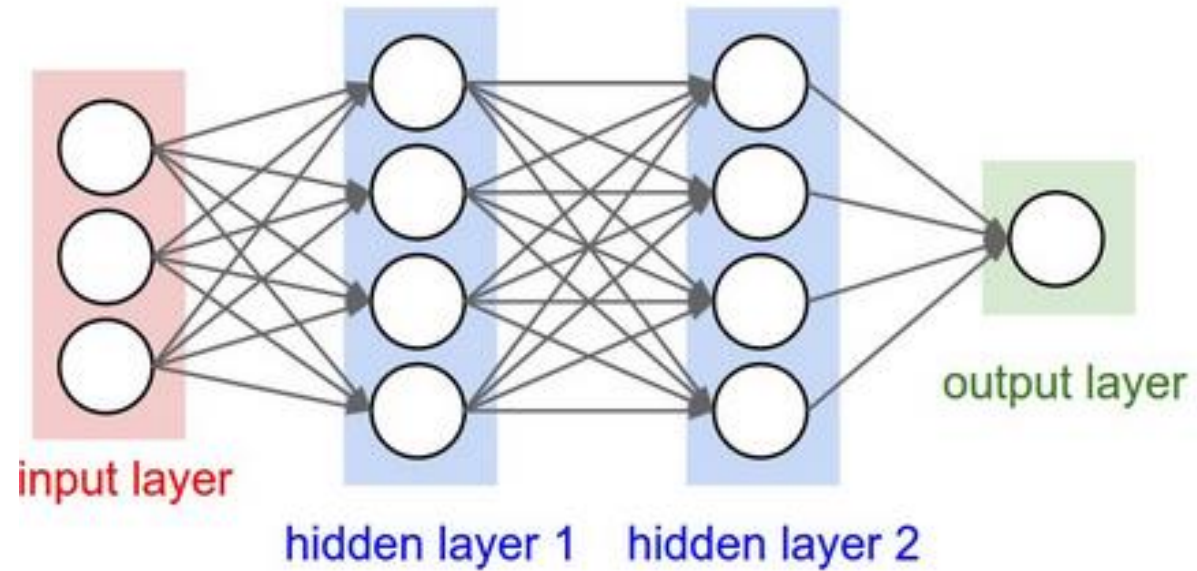
03

04

05

»» First approach,

Deep Learning Model



01

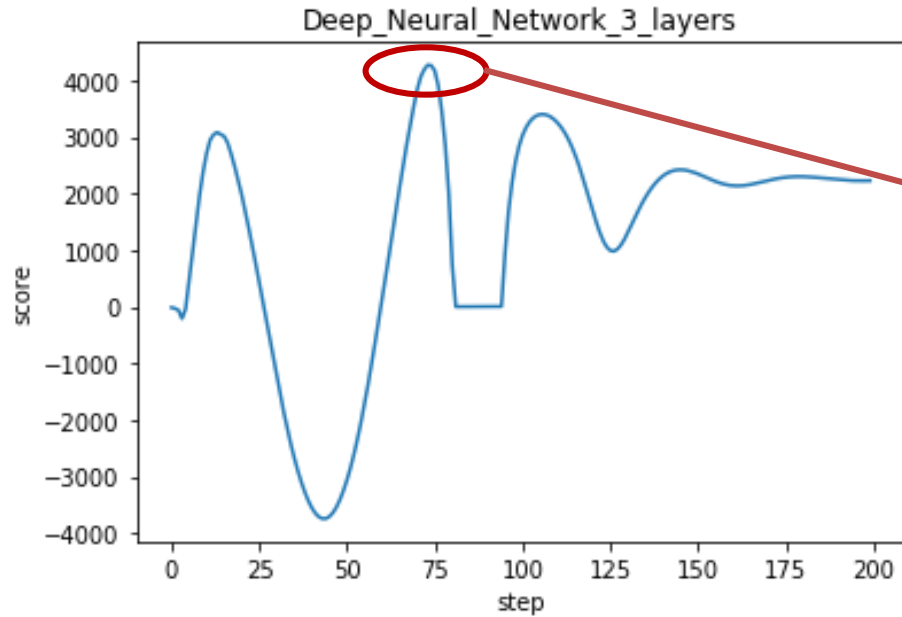
02

03

04

05

➤➤ DNN (Deep Neural Network)



acc_id	survival_time	amount_spent
7	13.37778132	0.98433147
15	13.37778132	0.98433147
16	13.37778132	0.98433147
18	13.37778132	0.98433147
19	13.37778132	0.98433147
22	13.37778132	0.98433147
24	13.37778132	0.98433147
28	13.37778132	0.98433147

Score를 가장 크게 한 step에서의 survival_time와 amount_spent은 모든 유저가 같은 값을 가졌으며, 리더보드 기준 4775.63점을 기록하였다.

01

02

03

04

05



다른 유저들의 생존시간과 결제금액을 맞추지 못하더라도,
하루에 돈을 많이 쓰고 빨리 이탈하는 유저를 잘 맞추는 것이 기대이익 증진에 도움이 된다!

01

02

03

04

05

»» Second approach,

Tree-based Model

정형 데이터 예측에서 최고의 성능을 보이는 Model

예측 성능 DNN과 다르게 변수 중요도 등을 출력하여 해석이 가능함

Hyperparameter에 따라 성능에 있어 민감하지만, 최근 빠르고 정확한 tuning 방법론들이 등장하면서 그 활용성이 더욱 커짐

01

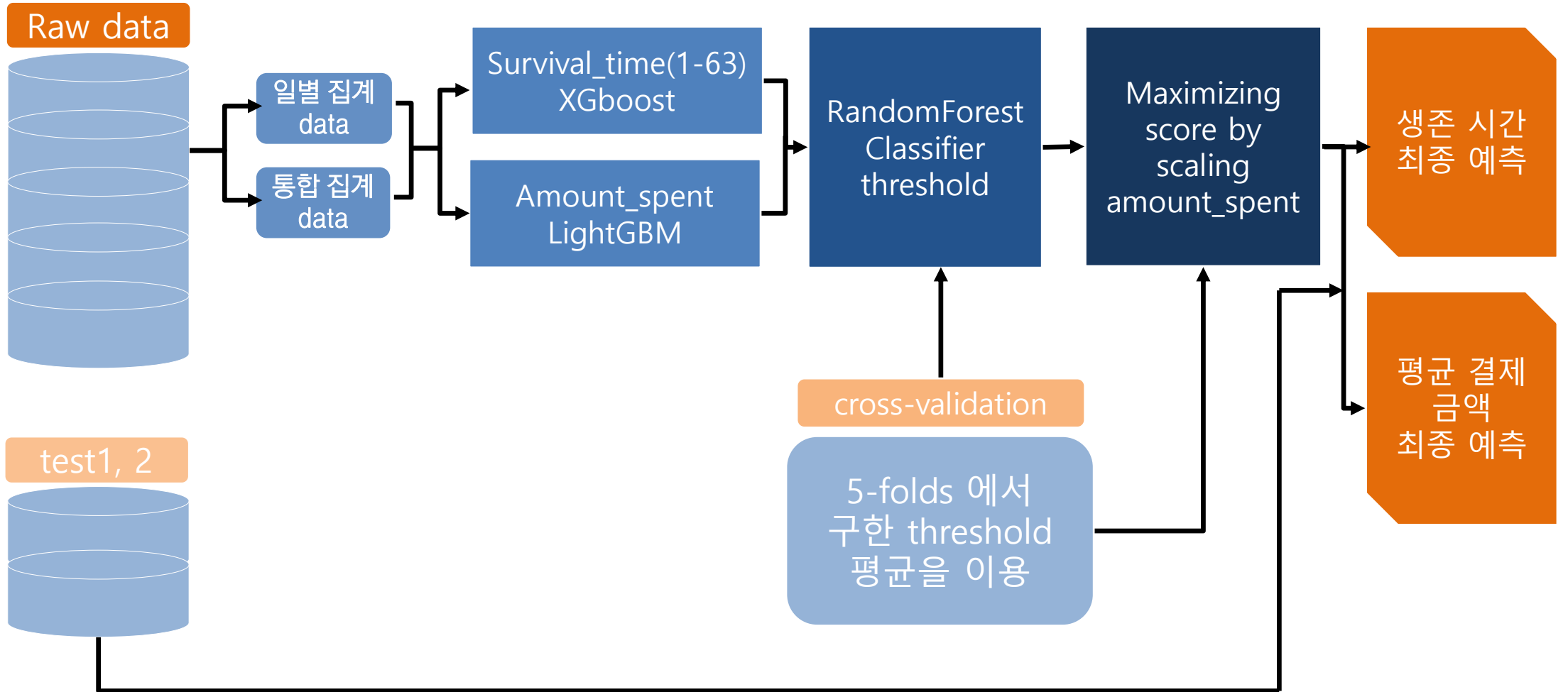
Modeling Pipeline

02

03

04

05




01

02

03

04

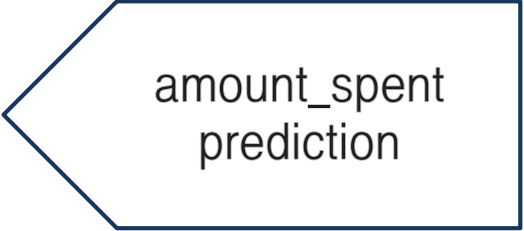
05



survival_time
prediction

모델 1. Survival Time regression
(XGBoost)

모델 3. 이탈 / 잔존 classification
(Random Forest)



amount_spent
prediction

모델 2. Amount Spent regression
(LightGBM)

모델 4. 과금 / 무과금 classification
(Random Forest)

01

02

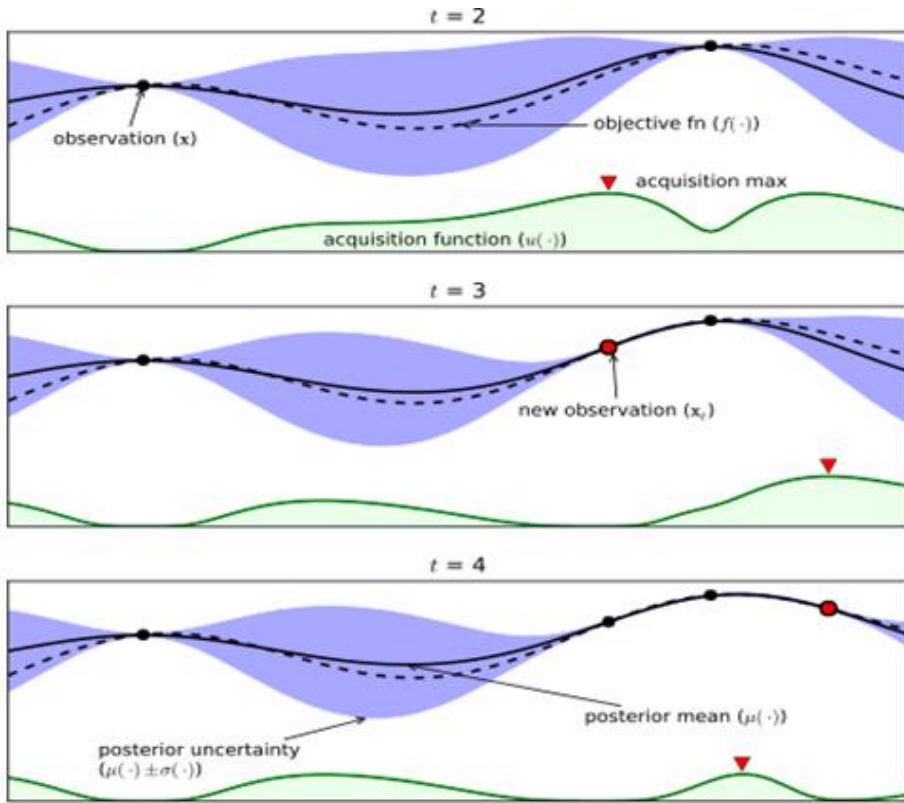
03

Hyper Parameter Tuning : Bayesian Optimization

조정할 수 있는 Tree Model에 대하여 베이지안 옵티마이저를 이용하여 최적화하였다

04

05



What is Bayesian Optimization?

어느 입력값 x 를 받는 미지의 Objective function f 를 상정하여, 그 함숫값 $f(x)$ 를 최대로 만드는 최적해 x^* 를 찾는 방법

목적 함수의 표현식을 명시적으로 알 지 못하면서, 하나의 함숫값 $f(x)$ 를 계산하는 데 오랜 시간이 소요되는 경우 효과적

가능한 한 적은 수의 입력값 후보들에 대해서만 그 함숫값을 순차적으로 조사하여, $f(x)$ 를 최대로 만드는 최적해 x^* 를 빠르고 효과적으로 찾는 것이 주요 목표

01

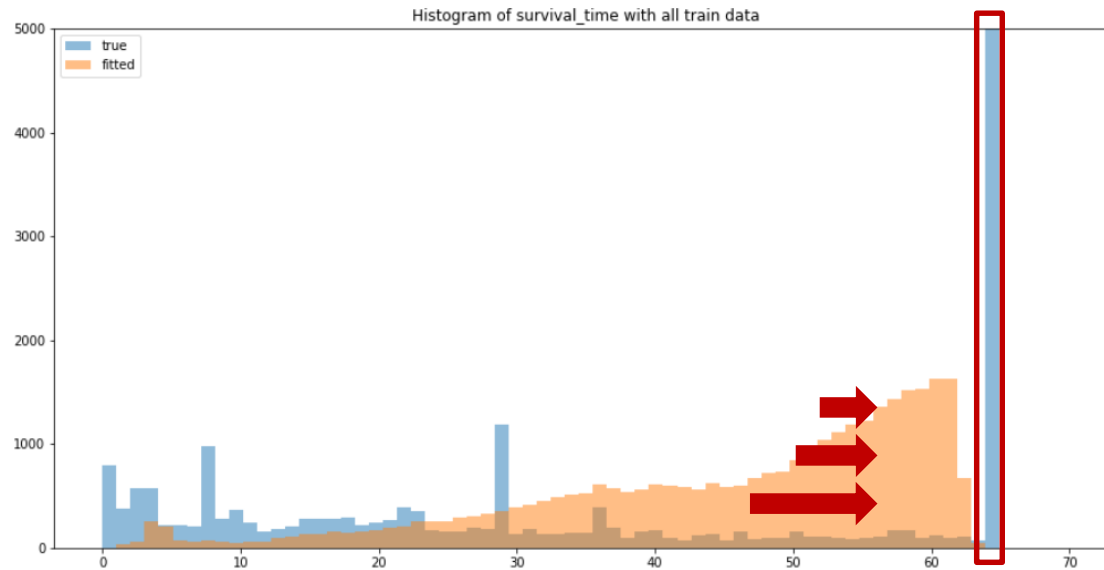
02

03

04

05

➤➤ 모델 1. survival_time regression



- 생존 시간(Survival_time)은 수치형 변수이긴 하지만, 다른 값들에 비해 '64'가 매우 많아, 이에 의한 예측값의 왜곡이 발생
→ 각 유저의 생존 시간을 과대 추정하게 되어 본 분석의 주요 목표 중 하나인 이탈 유저의 탐지를 어렵게 한다
- 머신러닝 모델은 MSE를 줄이는 방향으로 값을 예측하기 때문에, parameter tuning만으로는 이를 해결할 수 없다
- 학습을 진행할 유저에 대한 segmentation이 선행되어야 한다

01

➤➤ 모델 1. survival_time prediction : XGBoost

02

03

따라서, **이탈한 유저**(survival_time < 64) 의 정보 만을 활용하여 학습을 진행

04

05

- 주어진 score metric에서 가장 좋은 성능을 보이는 목적함수 'count:poisson' (Poisson Regression) 선택
- 여러 Boosting 모형에 해당 목적 함수가 있었지만, 같은 hyperparameter boundary 안에서 tuning 했을 때 제일 좋은 성능을 낸 XGBoost를 최종 예측 모형으로 채택
- 5-CV Bayesian optimization을 통해 최대한 많은 parameter를 동시에 tuning하여 모델의 성능 향상과 예측의 안정성을 확보하고자한다

01

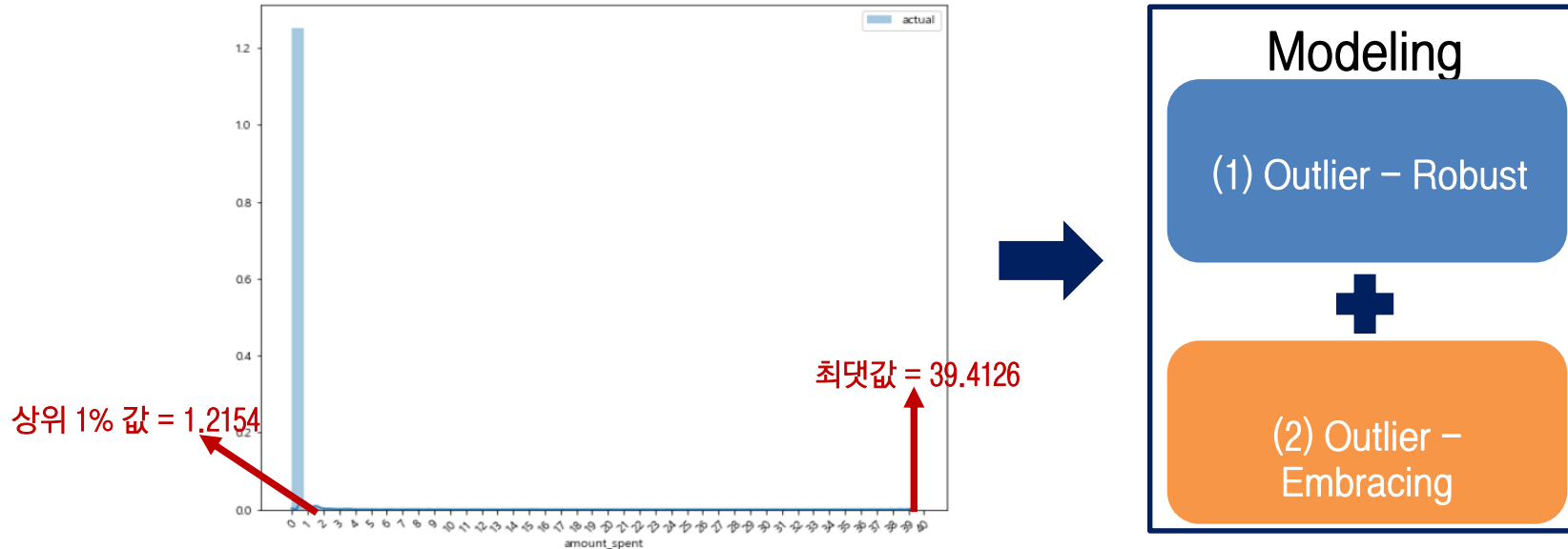
02

03

04

05

➤➤ 모델 2. amount_spent regression



(0, 1) 범위에 amount_spent 값이 약 99% 정도 분포하지만, 최댓값은 약 39로 outlier가 매우 심하게 나타나는 분포
일반 linear model을 적용할 시 예측값이 왜곡될 우려가 있으므로, outlier에 강건한 모형을 사용해야 한다
또한 기대 이익에 큰 플러스를 줄 수 있는 유저는 바로 outlier에 해당되는 유저이므로, 이들을 담아낼 수 있는 접근 역시 필요하다

01

➤➤ 모델 2. amount_spent regression : LightGBM

02

03

(1) Outlier-Robust Model

04

05

- Outlier에 Robust하여 전체적인 분포의 형태를 잘 맞춰줄 수 있는 Boosting model 적용
- Fitting 속도가 빨라 보다 많은 hyperparameter를 빠른 시간 내에 많이 tuning할 수 있으며, 정확도까지 보장하는 LightGBM model 채택
- XGBoost와 마찬가지로 5-CV Bayesian optimization algorithm 적용

01

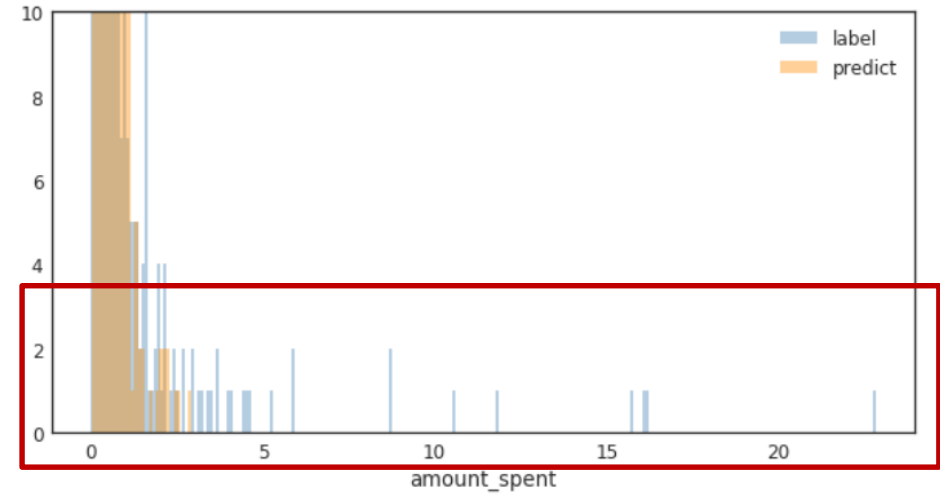
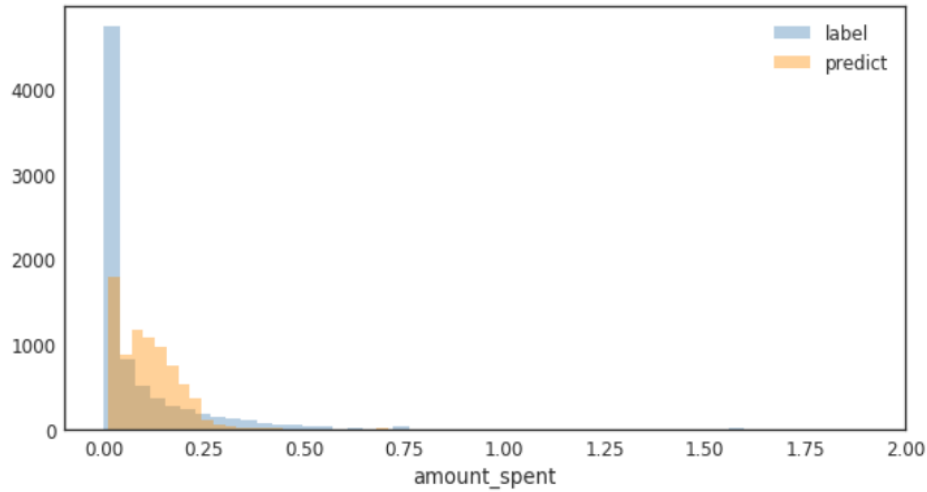
02

03

04

05

➤➤ 모델 2. amount_spent regression : LightGBM



LightGBM을 통해 예측한 amount_spent의 예측값과 실제값을 비교해봤을 때, 확실히 1보다 큰 'Outlier 유저' 에 대해서는 제대로 예측하지 못하는 경향을 보였다.

01

02

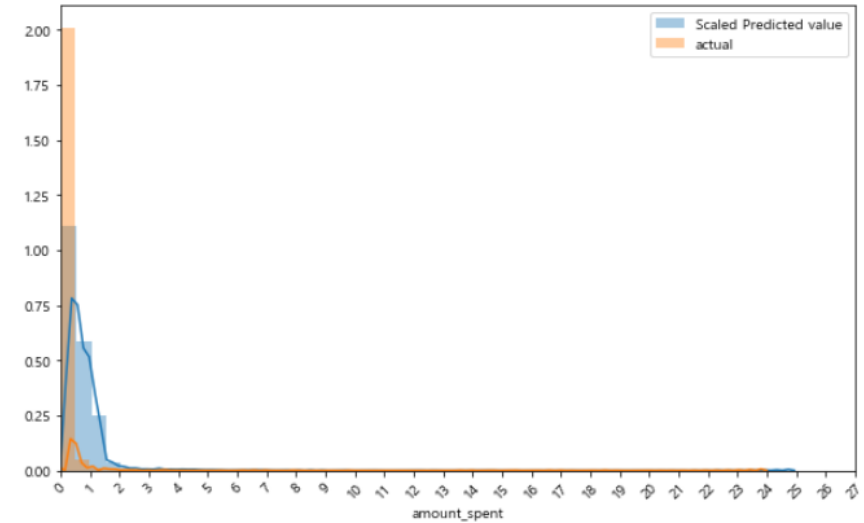
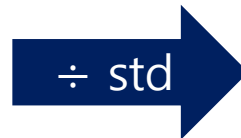
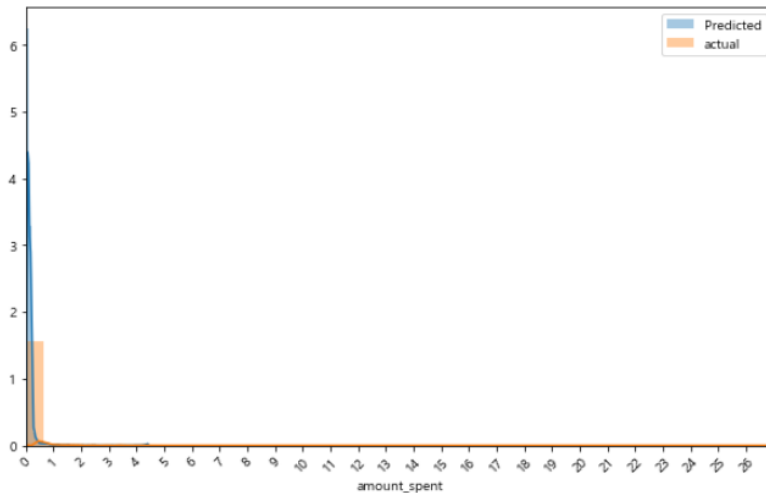
03

04

05

➤➤ 모델 2. amount_spent regression : Scaling

(2) Outlier-Embracing scaling



- 오차를 최소화하는데 초점이 맞춰진 기존 머신러닝 방법론으로는 기대 이익에 결정적인 역할을 하는 ‘Outlier 유저’ 를 잡아내는 데 한계 존재
- 기존 LightGBM의 예측값을 ‘예측값의 표준편차’ 로 나누어 주는 **Scaling step** 도입
- 그 결과 예측값과 실제 분포의 모양이 더욱 유사해졌으며, score도 획기적으로 높일 수 있었다

01

02

03

04

05

»» 모델 3. 이탈 / 잔존 classification

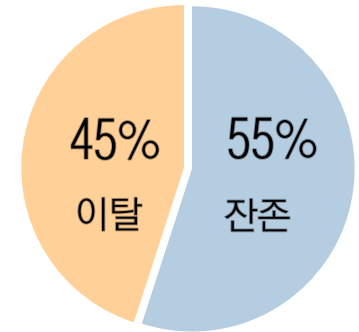
모델 1로 survival_time을 예측할 때,

실제 잔존고객임에도 불구하고 tree model의 한계로 인하여 tail값인 64로는 아예 예측하지 않는 문제가 발생하였다

이러한 문제를 해결하기 위하여

이탈 고객(survival_time = 1~63) vs 잔존 고객(survival_time = 64) 으로

binary classification을 진행한 후,



이 때의 분류결과가

잔존이면 survival_time → 64

이탈이면 survival_time → 모델 1의 predict 결과

01

02

03

04

05

» survival_time prediction 결과 예시

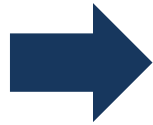
모델 1
Survival time
Prediction

acc_id	survival_time
1	54.234
2	63.678
3	20.343
...	...

모델 3
이탈 / 생존
Classification

acc_id	이탈 확률	생존 확률	생존 여부
1	0.32	0.68	생존
2	0.03	0.97	생존
3	0.90	0.10	이탈
...

Survival Time
예측 결과



acc_id	survival_time	이탈 확률	생존 확률	생존 여부
1	64	0.32	0.68	생존
2	64	0.03	0.97	생존
3	20.343	0.90	0.10	이탈
...

01

02

03

04

05

➤➤ 모델 4. 과금 / 무과금 classification

Survival time과 마찬가지로 모델 3으로 amount_spent값을 예측할 때,
실제 무과금고객임에도 불구하고 tree model의 한계로 인하여 tail값인 0으로는 거의 예측하지 않는 문제가 발생하였다

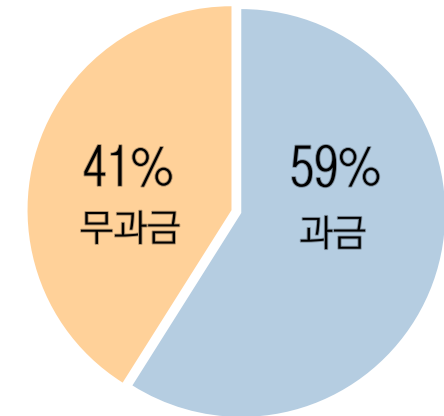
따라서,

과금($\text{amount_spent} > 0$) vs 무과금($\text{amount_spent} = 0$) 으로
binary classification을 진행한 후,

이 때의 분류결과가

무과금이면 amount_spent $\rightarrow 0$

과금이면 amount_spent \rightarrow 모델 3의 predict 결과



01

02

03

04

05

≫≫ amount_spent prediction 결과 예시

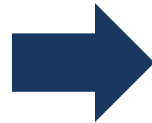
모델 2
Amount spent
Prediction

acc_id	amount_spent
1	0.0000
2	0.0056
3	1.4285
...	...

모델 4
과금 / 무과금
Classification

acc_id	과금 확률	무과금 확률	과금 여부
1	0.02	0.98	무과금
2	0.30	0.70	무과금
3	0.10	0.90	과금
...

Amount Spent
예측 결과



acc_id	amount_spent	과금 확률	무과금 확률	과금 여부
1	0	0.02	0.98	무과금
2	0	0.30	0.70	무과금
3	1.4285	0.10	0.90	과금
...

01

»» 예측결과

02

그러나 모델 3와 모델 4에서 분류한 결과를 그대로 적용하면 오히려 score가 떨어지는 결과를 보였다

03

04

Why?

05

Score function의 특성상,

Survival Time

유저가 생존할 것이라고 예측시 기대이익이 0이 되어버리는 문제 발생

→ **확실하게 생존할 유저만** 생존으로 예측해야함

→ **생존이라고 분류할 확률이 높은 유저만** 생존으로 판단하고 예측값을 64로 보냄

Amount Spent

유저가 과금하지 않을 것이라고 예측시 기대이익이 0이 되어버리는 문제 발생

→ **확실하게 과금하지 않을 유저만** 무과금으로 예측해야함

→ **무과금이라고 분류할 확률이 높은 유저만** 예측값을 0으로 보냄

01

➤➤ 모델 3 . 이탈 / 잔존 classification & 모델 4 . 과금 / 무과금 classification

02

03

RandomForest

04

다양한 모델 중 가장 높은 자체평가점수를 얻은 RandomForest를 선택하였다

05

앞에서 말했듯이,

확실하게 생존할 유저와 확실하게 돈을 쓰지않을 유저를 분류할 때 확률임계값(threshold)을 높여주어야 한다

➔ Grid search를 통해 score를 최대로 하는 **최적의 생존 확률의 threshold와 무과금 확률의 threshold 조합**을 찾기

01

02

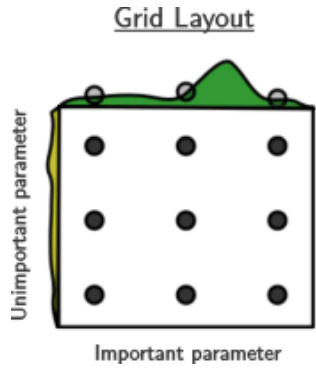
03

04

05

→ 최적의 생존 확률의 threshold와 무과금 확률의 threshold 조합

예시



Best 생존확률 threshold → 0.8

Best 무과금확률 threshold → 0.9

Survival Time
예측 결과

acc_id	survival_time	이탈 확률	생존 확률	생존 여부
1	54.234	0.32	0.68	이탈
2	64	0.03	0.97	생존
3	20.343	0.90	0.10	이탈
...

← 생존확률이 0.8 이상



Amount Spent
예측 결과

acc_id	amount_spent	과금 확률	무과금 확률	과금 여부
1	0	0.02	0.98	무과금
2	0.0056	0.30	0.70	무과금
3	1.4285	0.10	0.90	과금
...

← 무과금확률이 0.9 이상

01

02

03

04

05

즉, acc_id 기준으로 예측값을 종합해보면

acc_id	survival_time	이탈 확률	생존 확률	생존 여부
1	54.234	0.32	0.68	이탈
2	63.678	0.03	0.97	생존
3	20.343	0.90	0.10	이탈
...

acc_id	amount_spent	과금 확률	무과금 확률	과금 여부
1	0.0000	0.02	0.98	무과금
2	0.0056	0.30	0.70	과금
3	1.4285	0.10	0.90	과금
...



acc_id	survival_time	amount_spent
1	54.234	0
2	64	0.0056
3	20.343	1.4285
...

01

➤➤ amount_spent adjusting by score function

02

03

04

- 기대 이익(Score) 최적화의 핵심은

05

‘많은 금액을 결제하는 유저의 결제 금액을 정확히 예측하는 것’

- 특히 상위 1~3% 이내의 결제 금액은 대부분의 결제 금액보다 훨씬 클 뿐만 아니라, 변동성도 매우 크다
- 따라서 amount_spent의 예측값이 **‘몇 이상(threshold) 몇 이하(ceiling) 값일 때 몇 배(scale)를 곱한다’** 는 adjustment를 모델링의 마지막 과정으로 수행
- adjustment에 필요한 3가지의 parameter 는 Grid search를 통해 score를 최대화하는 것으로 채택한다

01

02

03

04

05

» amount_spent adjusting by score function

```
pred_df2['amount_spent'].map(lambda x: scale*x if x >= ths and x <= ceil else x)
```

'몇 배' '몇 이상' '몇 이하'

Grid Search

```
19 th scale: 3.0 / 32 th amount_spent threshold: 4.2 / 9 th amount_spent ceilings: 15.528607821434065  
/ Score is: 7799.522122916803  
  
The best scale, threshold and ceiling are: (scale: 3.0 / threshold: 1.6 / ceiling: 15.528607821434065)  
(7931.942652578404,  
'(scale: 3.0 / threshold: 1.6 / ceiling: 15.528607821434065)')
```

< Grid Search 과정 예시 >

05. 사후분석

01

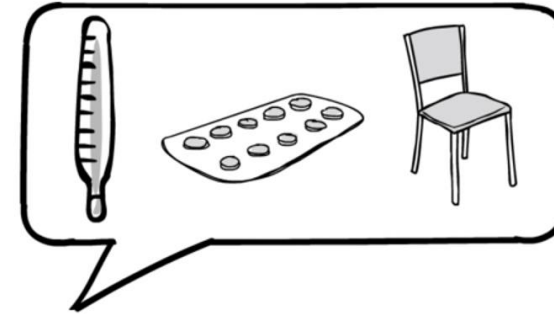
02

03

04

05

LOCAL
INTERPRETABLE
MODEL-AGNOSTIC
EXPLANATIONS



01

02

03

04

05

LIME

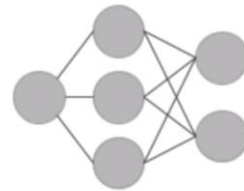
: 어떤 예측 모델이든 그 모델이 왜 그렇게 예측했는지 설명해주는 모델

관심 변수의 값을 중심으로 임의의 값을 생성한 뒤, 그에 따른 예측 값이 바뀌는 정도를 확인하여 해당 변수의 중요도를 파악

라임은 어떤 예측 모델이더라도 적용 가능하기 때문에, 딥러닝 혹은 복잡한 머신 러닝 모델 설명 시 사용 가능

앞서 사용한 4 종류의 모델에 LIME을 적용

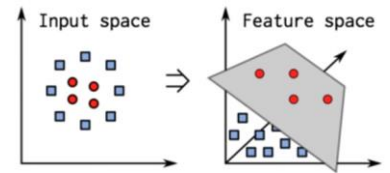
Neural Networks



Random Forests

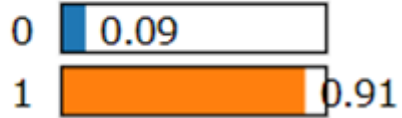


SVM with any kernel



-> 실제 $survival_yn = 1$ 인 observation을 1로 예측한 경우

Prediction probabilities



Feature Value

num_of_days	28.00
pledge_days	28.00
sum_of_npckill	0.00
sum_of_playtime	90.57
level_up_rate	0.00

0

1

sum_of_npckill <= 0.98

- 0.02 27 < num_of_days <= 28
- 0.02 25 < pledge_days <= 28
- 0.01 sum_of_playtime > 77.87
- 0.01 level_up_rate <= 0

- ('27.00 < num_of_days <= 28.00', 0.020877388057330328)
- ('25.00 < pledge_days <= 28.00', 0.017796451557984077)
- ('sum_of_npckill <= 0.98', -0.01758699842664315)
- ('sum_of_playtime > 77.87', 0.0122524491067536)
- ('level_up_rate <= 0.00', 0.01159283103090177)

01

02

03

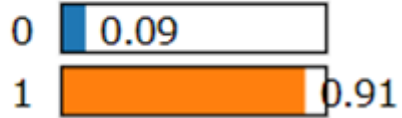
04

05

01
02
03
04
05

-> 실제 $survival_yn = 1$ 인 observation을 1로 예측한 경우

Prediction probabilities



Feature Value

num_of_days	28.00
pledge_days	28.00
sum_of_npckill	0.00
sum_of_playtime	90.57
level_up_rate	0.00

0

1

sum_of_npckill <= 0.98

- 0.02 27 < num_of_days <= 28
- 0.02 25 < pledge_days <= 28
- 0.01 sum_of_playtime > 77.87
- 0.01 level_up_rate <= 0

- ('27.00 < num of days <= 28.00', 0.020877388057330328)
- ('25.00 < pledge days <= 28.00', 0.017796451557984077)
- ('sum_of_npckill <= 0.98', -0.01758699842664315)
- ('sum_of_playtime > 77.87', 0.0122524491067536)
- ('level_up_rate <= 0.00', 0.01159283103090177)

01

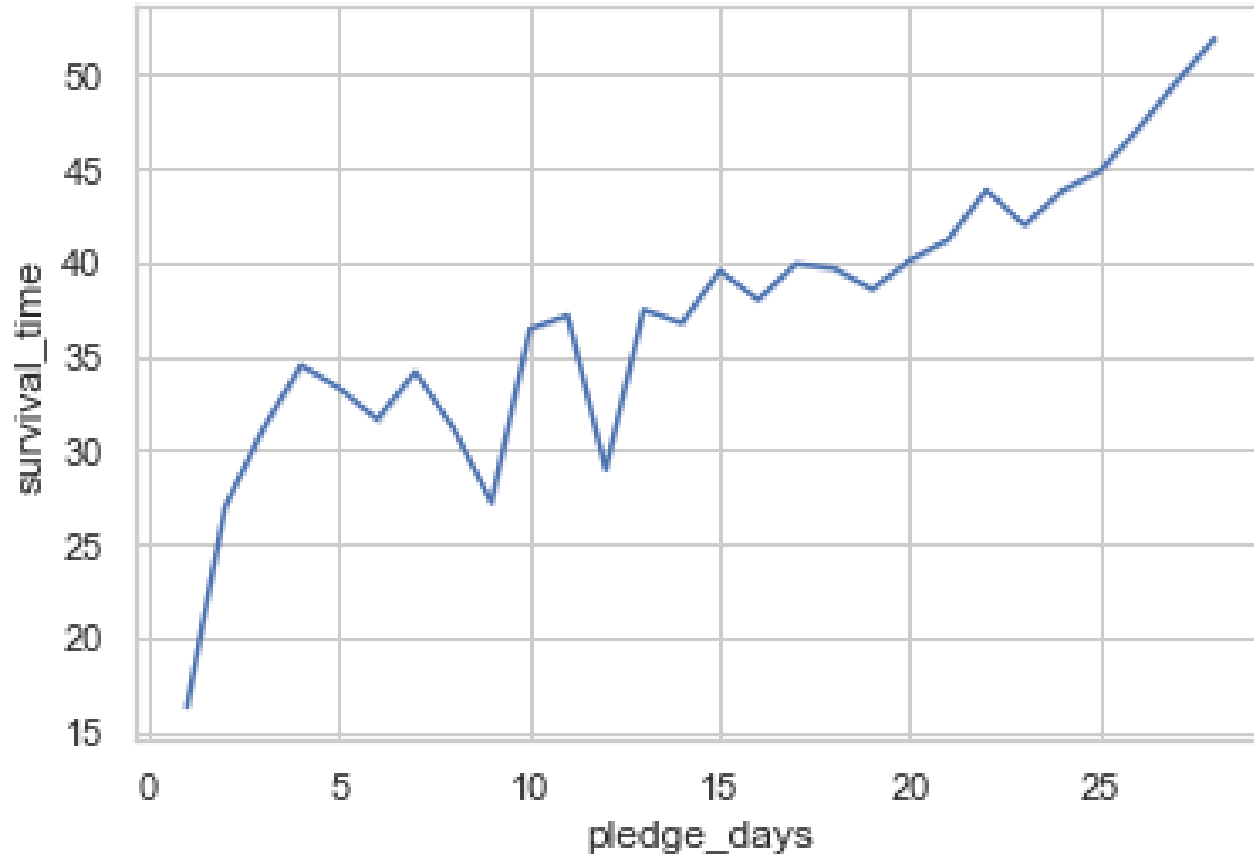
02

03

04

05

혈맹 활동에 얼마나 참여하는가



01

02

03

04

05

혈맹 활동

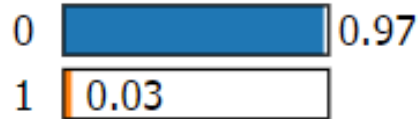
기간 내에 pledge 활동일이 많을수록 survival_time 증가

→ 혈맹 활동 지원을 통한 추가 생존 기간 증가

“게임 내 사회활동이 지속적인 플레이의 원동력”

-> 실제 $amount_yn = 0$ 인 observation을 0으로 예측한 경우

Prediction probabilities

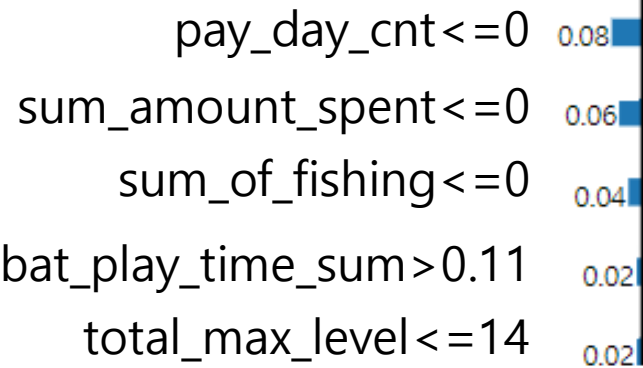


Feature Value

pay_day_cnt	0.00
sum_amount_spent	0.00
sum_of_fishing	0.00
non_combat_play_time_sum	18.81
total_max_level	2.00

0

1



- ('pay_day_cnt <= 0.00', -0.07904589126085286)
- ('sum_amount_spent <= 0.00', -0.060467484029733926)
- ('sum_of_fishing <= 0.00', -0.03727603810009532)
- ('non_combat_play_time_sum > 0.11', -0.016549714318445274)
- ('total_max_level <= 14.00', -0.016257129785629695)

01

02

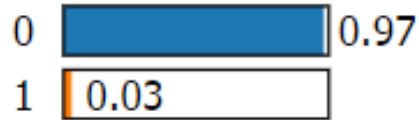
03

04

05

-> 실제 amount_yn = 0 인 observation을 0으로 예측한 경우

Prediction probabilities

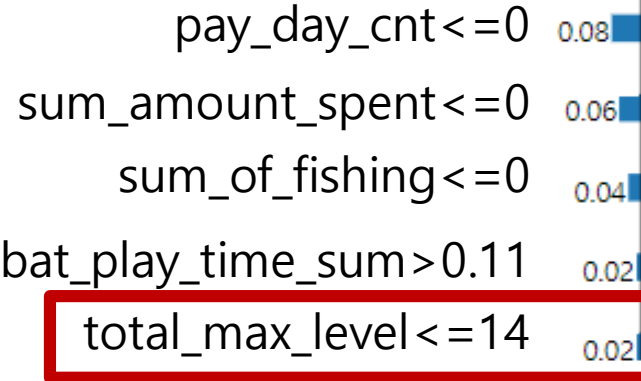


Feature Value

pay_day_cnt	0.00
sum_amount_spent	0.00
sum_of_fishing	0.00
non_combat_play_time_sum	18.81
total_max_level	2.00

0

1



- ('pay_day_cnt <= 0.00', -0.07904589126085286)
- ('sum_amount_spent <= 0.00', -0.060467484029733926)
- ('sum_of_fishing <= 0.00', -0.03727603810009532)
- ('non combat play time sum > 0.11', -0.016549714318445274)
- ('total_max_level <= 14.00', -0.016257129785629695)

01
02
03
04
05

01

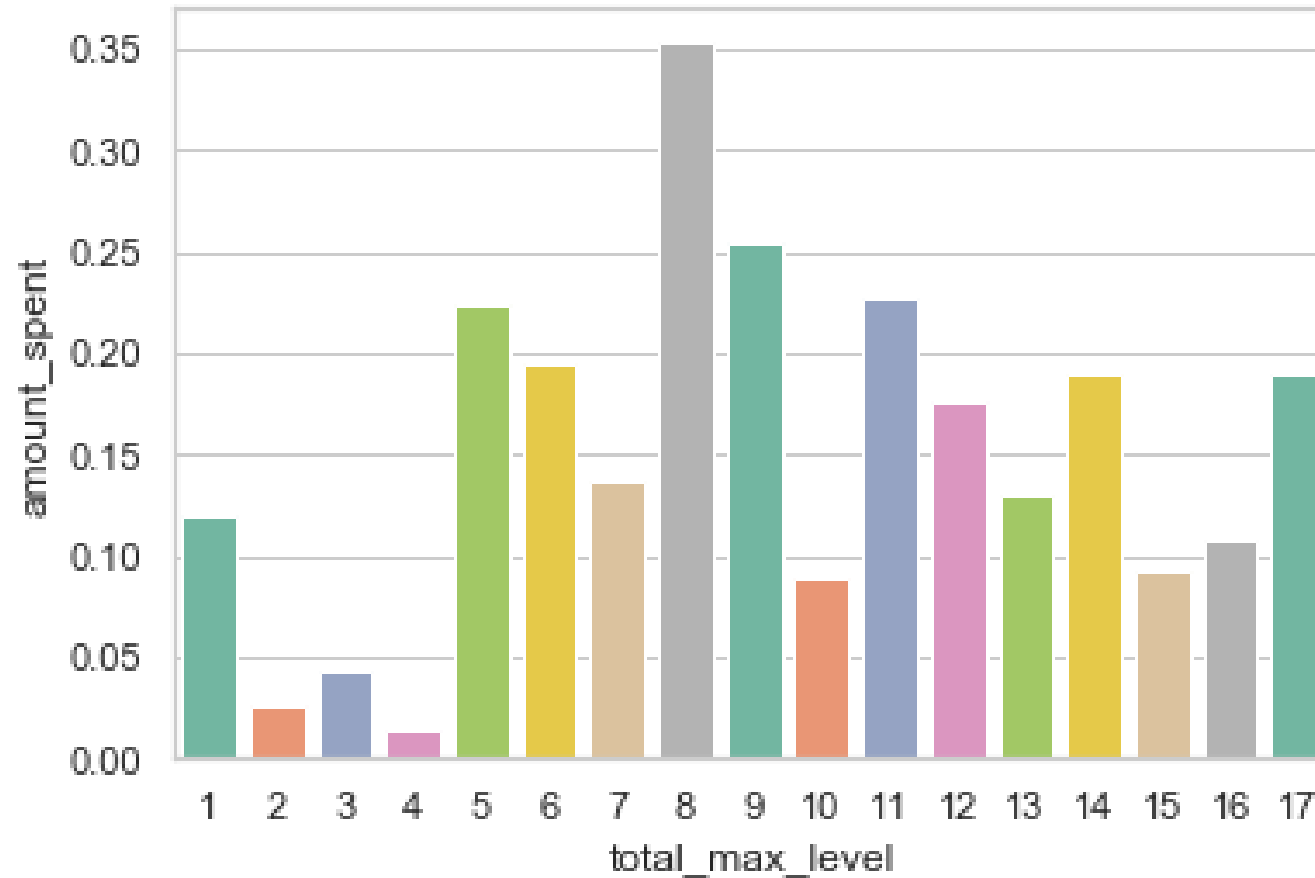
02

03

04

05

본캐의 레벨이 어느정도 되는가



01

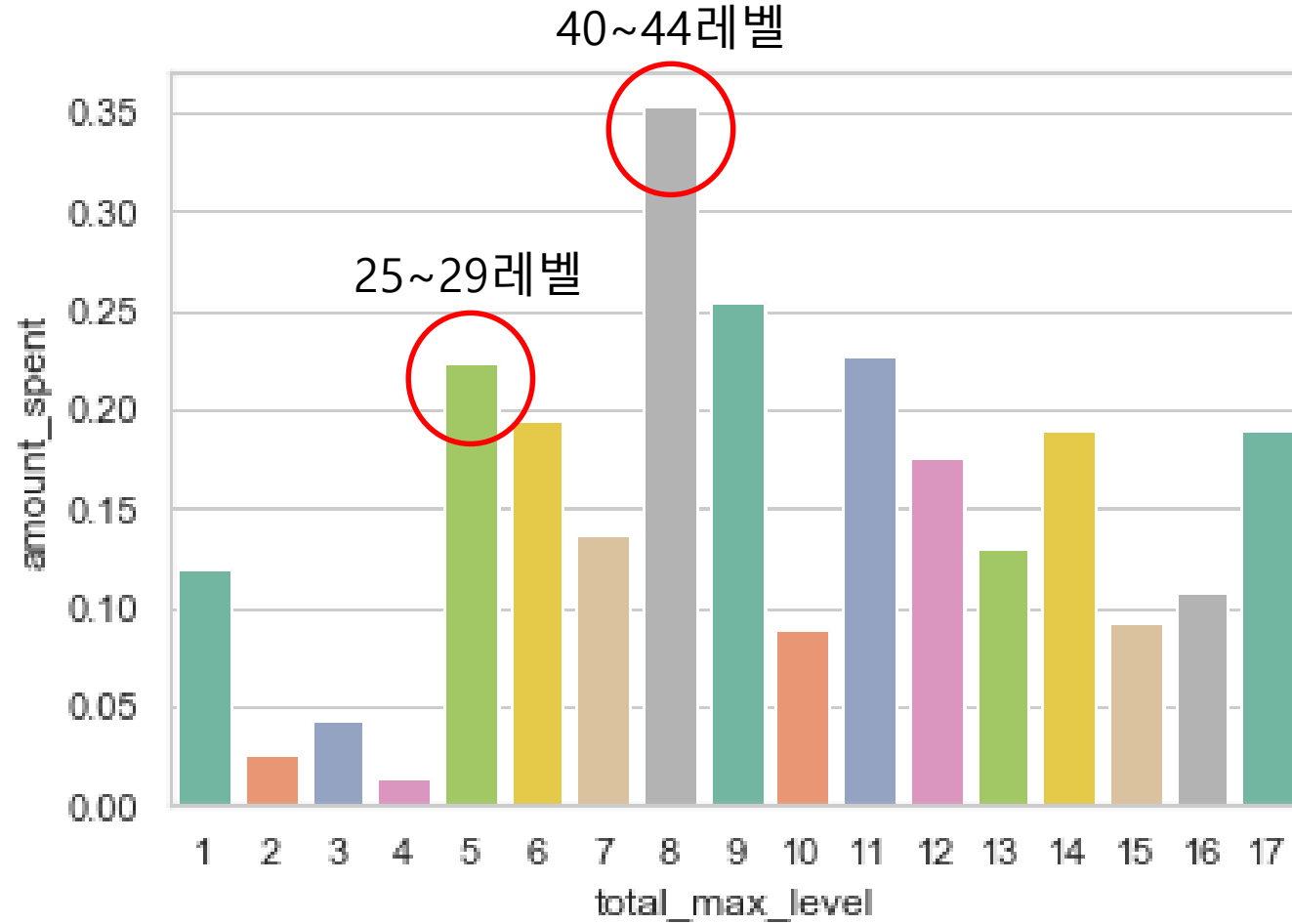
02

03

04

05

본캐의 레벨이 어느정도 되는가



01

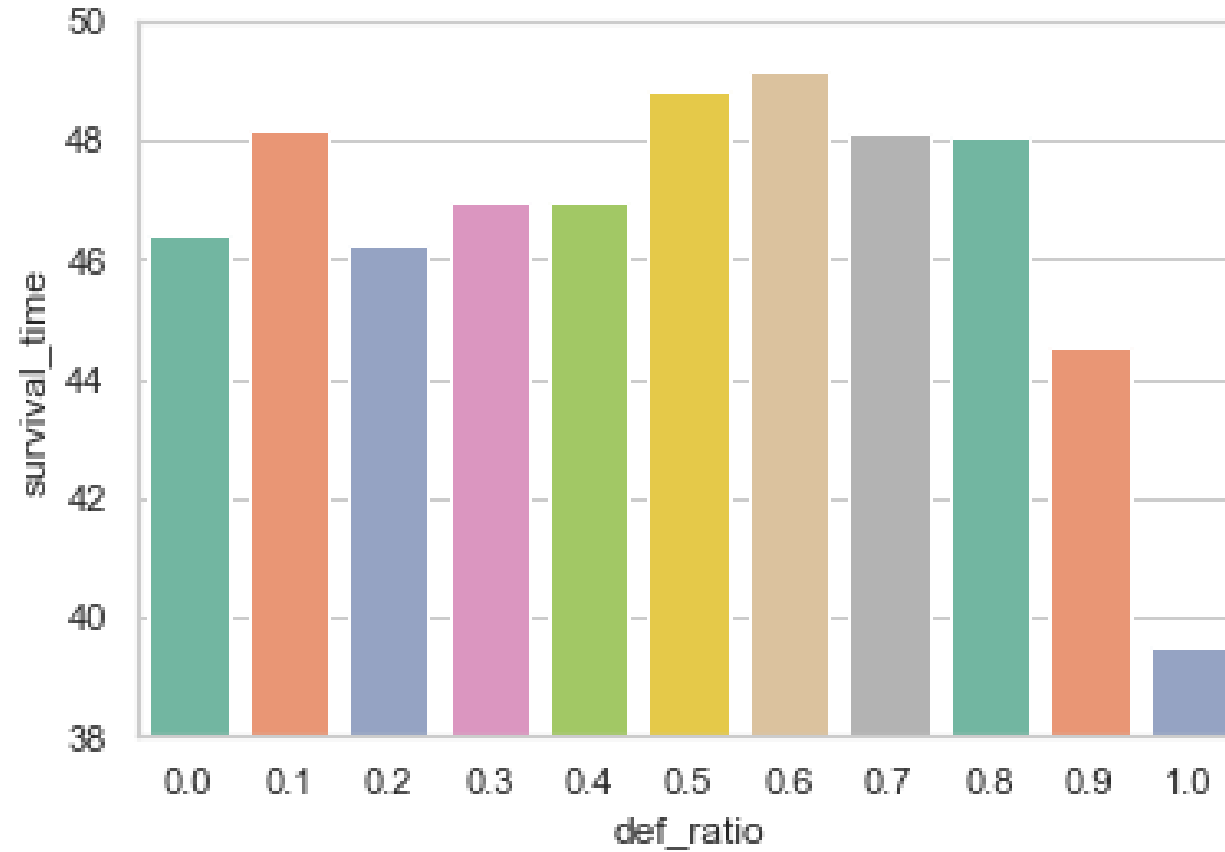
02

03

04

05

막피 이슈



01

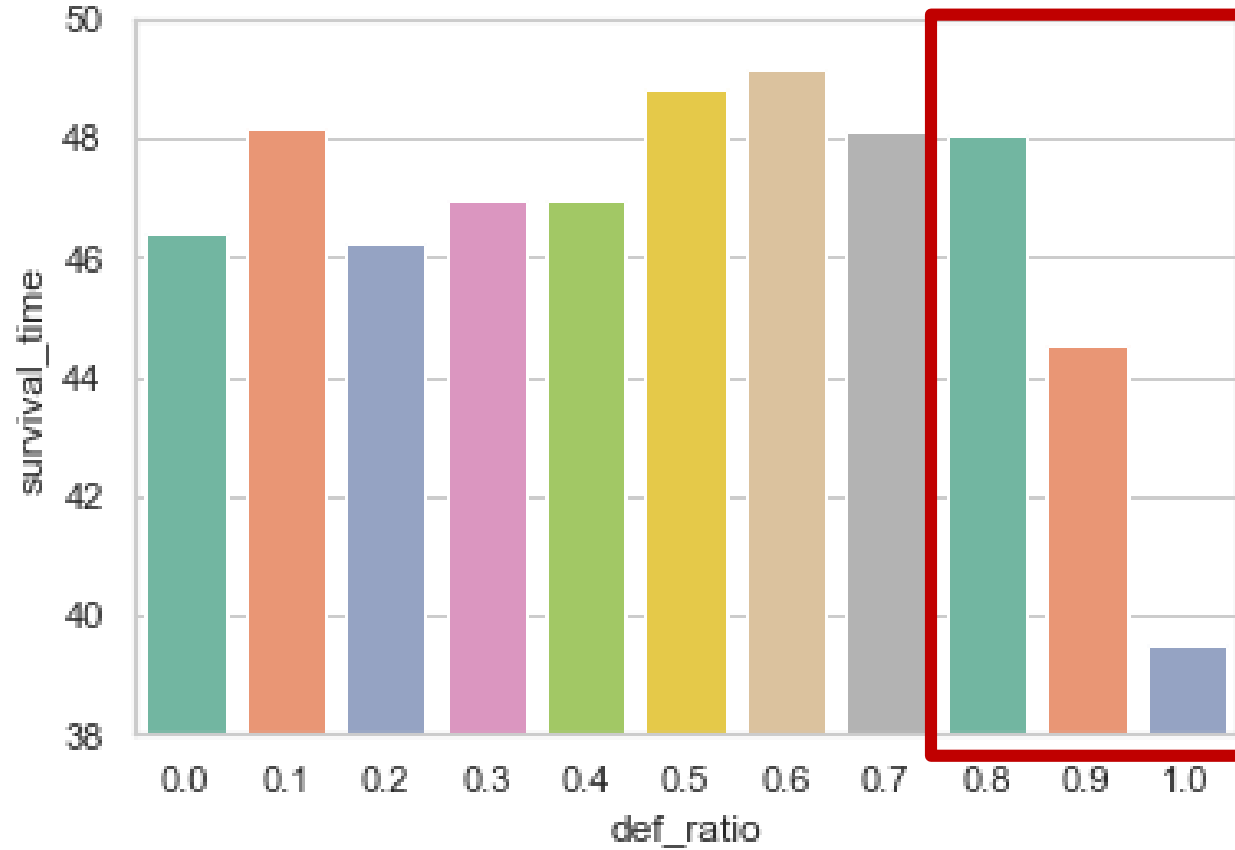
02

03

04

05

막피 이슈



01

02

03

04

05

막피 이슈

막피를 가한 횟수에 비하여 막피를 당한 횟수의 비율이 높은 경우,
survival_time이 급격하게 낮아지는 것을 확인

무차별적인 PK로 인하여 게임 플레이에 지장이 생기고,
이로 인하여 이탈하는 플레이어들이 많음

01

02

03

04

05

기대 이익 최대화를 위한 방안

육성 가이드라인 개편

특정 레벨 유저에 대한 맞춤 프로모션

막피 이슈 개선

2019 BigContest

Thank you

우리팀 화이팅